09/707,462

WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Thursday, March 25, 2004

Hide?	Set Name	Query	Hit Count
	DB=USPT; PLUR=NO; OP=ADJ		
☐	L11	L8 same record$1 same key	2
☐	L10	L8 same record$1 same index same key	0
☐	L9	L8 same record$1 same index same key same first	0
☐	L8	cyclical adj1 redundancy adj1 check	768
☐	L7	L6 and l4	4
☐	L6	l2 and l1	49
☐	L5	L4 and l3 and l2 and l1	0
☐	L4	(key near7 record) same value	873
☐	L3	cataloging	1120
☐	L2	((cyclical adj1 redundancy adj1 check) or CRC-CCITT or CRC-16 or CRC-32 or CRC)	28989
☐	L1	(707/1 or 707/100).ccls.	2690

END OF SEARCH HISTORY

WEST

Generate Collection Print

L27: Entry 1 of 8 File: USPT Apr 22, 2003

DOCUMENT-IDENTIFIER: US 6553372 B1
TITLE: Natural language information retrieval system

Abstract Text (1):
A natural language information retrieval (NLIR) system employing a hash table technique to reduce memory requirements and a proxy process module to improve processing speed on multi-processor platforms. The NLIR system includes a Dynamic Link Library (DLL) search engine annex that implements a number of improvements that allow the preexisting natural language processing (NLP) core code module to operate sufficiently fast in a limited-memory environment. The improvements relate to (1) reducing storage requirements, (2) increasing processing speed, (3) improved operation on multi-processor platforms, and (4) a trouble-shooting mechanism. The NLIR system includes three modes of operation. First, during index processing, the NLIR system prepares documents for NLP searching to create a group of searchable documents. Second, during question processing, the NLIR system receives a natural language question and, for each document in the group of searchable documents, computes a document score connoting the likelihood that the document includes an answer to the natural language question. Third, during debugging, the NLIR system receives trouble-shooting requests and returns diagnostic reports, such as a document trace report and a question trace report.

Brief Summary Text (2):
The present invention relates generally to the field of computer software and, more particularly, to a natural language information retrieval system employing a hash table technique to reduce memory requirements, a proxy process module to improve processing speed on multi-processor computing platforms, and a debugging module that is not shipped along with the natural language information retrieval system.

Brief Summary Text (16):
The present invention meets the needs described above in an NLIR utility that stores LFTs using a hash-table technique that relies on a quasi-random hash value computed for each LFT. During index processing, the NLIR utility computes hash values for each LFT present in a document. The hash value is parsed into an address hash and a signature hash, and each LFT is represented by its hash signature stored in an array at a memory location based on the associated address hash. The NLIR utility uses this technique to create a hash-table fingerprint for each document in a group of searchable documents. Each fingerprint, which includes a representation of the LFTs in the corresponding document, is stored in a relatively small hash-table array.

Brief Summary Text (17):
During question processing, the NLIR utility obtains LFTs for a natural language question on the fly, and computes hash values for the question LFTs using the same formula that was used during index processing. The NLIR utility then compares the hash values for the question LFTs to the hash-table fingerprints stored for each document in the group of searchable documents. A match between a hash value for a question LFT and a hash value found in a searched fingerprint indicates a very high likelihood that the corresponding document contains an LFT matching the question LFT. The NLIR utility assigns a predefined score to each matching LFT based on the type of LFT, and sums the scores to produce a document score for each document. The NLIR utility returns the document scores to a search engine, which displays the documents in a priory order based on the document scores returned by the NLIR utility.

Brief Summary Text (18):

Thus, during index processing, the NLIR utility preprocess the group of searchable documents to create a hash-table fingerprint for each document without having to store the actual LFTs for each document. Then, during question processing, the NLIR utility performs LFT comparisons directly on the hash-table fingerprints without having to generate the actual LFTs for the searched documents. This use of hash tables gives the NLIR utility the combination of acceptable processing speed and acceptable memory requirements when implemented in connection with a storage-limited program module, such as a CD-ROM title. That is, the NLIR utility does not require on-the-fly LFT processing or storage of a complete LFT listing for each document searched. The resulting NLIR utility may also be used in connection with engines for searching relatively large-scale distributed computing environments, such as search engines used in connection with local-area networks, wide-area networks, intranets, the Internet, and so forth

Brief Summary Text (23):
More specifically, the invention provides an NLIR utility configured to implement a method for creating a group of searchable documents, which is also referred to as "index processing." For each document, the NLIR utility receives text defining the document and parses the text into a plurality of text portions, such as sentences. The NLIR utility obtains one or more logical form relationships corresponding to each text portion, typically by passing the text portion to a conventional NLP core code module. Once logical form relationships have been obtained for the entire document, the NLIR utility defines an array having a size corresponding to the number of logical form relationships for the document. The NLIR utility then creates a hash-table fingerprint for the document by computing a hash value for each logical form relationship. For each hash value, the NLIR utility obtains an address hash and a signature hash based on the corresponding hash value and stores the signature hash in the array at a memory location corresponding to the address hash.

Brief Summary Text (24):
The NLIR utility may parse each hash value to obtain the corresponding address hash and signature hash. The NLIR utility may also identify an array index for an array entry point corresponding to the address hash. If the array entry point is empty, the NLIR utility may store the signature hash at the array entry point. Alternatively, if the array entry point is not empty, the NLIR utility may increment the array index of the array entry point until an empty memory location is defined and store the signature hash at the empty memory location.

Brief Summary Text (25):
More specifically, the NLIR utility may set the array index for the array entry point to the remainder of the address hash divided by the size of the array. In addition, the NLIR utility may set the size of the array to a predetermined percentage larger than the number of logical form relationships for the document. For example, the predetermined percentage may be 110%, the hash value may be a 32-bit value, the address hash may be the upper 16 bits of the hash value, and the signature hash may be the lower 19 bits of the hash value.

Brief Summary Text (26):
The NLIR utility is also configured to respond to a natural language question, which is also referred to as "question processing." During question processing, the NLIR utility receives a natural language question and obtains one or more logical form relationships for the question, typically by passing the question to the NLP core code module. Upon obtaining the question logical form relationships, the NLIR utility computes a hash value corresponding to each logical form relationship for the question. Then, for one or more document in the group of searchable documents, the NLIR utility compares the hash values corresponding to the logical form relationships for the question to the hash-table fingerprint for the document, and identifies one or more matching hash values.

Brief Summary Text (27):
The NLIR utility may also obtain a score for each matching hash value and, in response, sums the scores to compute a document score for each document connoting the likelihood that the document contains an answer to the natural language. The NLIR utility may then pass the document scores to a search engine that ranks the documents in order of their respective document scores. The search engine can display a list of

highest-ranking documents as a suggestion list of documents that likely contain an answer to the natural language question.

Brief Summary Text (28):
During question processing, the NLIR utility may parse a current hash value into a current address hash and a current signature hash. Parsing the hash value means that the NLIR utility may utilize a first subset of the hash value and the address hash a second subset of the hash value as the signature hash. These subsets may or may not overlap, and may or may not contain all of the digits of the hash value. The NLIR utility may then identify an array entry point in the array corresponding to the current address hash. If the array entry point is not empty, the NLIR utility may identify one or more consecutively-addressed data-containing memory locations beginning with the array entry point.

Brief Summary Text (29):
The NLIR utility then compares the current signature hash to the data value stored at each of the consecutively-addressed data-containing memory locations. If the current signature hash matches the data value stored in any of the consecutively-addressed data-containing memory locations, the NLIR utility identifies the current hash value as a matching hash value. Alternatively, if the array entry point is empty, the NLIR utility may identify the current hash value as a non-matching hash value. In addition, if the current signature hash does not match the data value stored at any of the consecutively-addressed data-containing memory locations, the NLIR utility may identify the current hash value as a non-matching hash value.

Brief Summary Text (30):
The invention also provides an NLIR system that includes an NLIR module configured for creating a group of searchable documents. For each document, the NLIR module receives text defining the document from a search engine and returns a hash-table fingerprint including a representation of logical form relationships for the document to the search engine. In addition, for each document, the NLIR module receives a natural language question and the hash-table fingerprint for the document from the search engine. In response, the NLIR module returns a document score to the search engine connoting the likelihood that the document contains an answer to the natural language question. The NLIR system may also include a search engine configured for ranking the documents in order of their respective document scores. The search engine may also display a list of highest-ranking documents as a suggestion list of documents containing an answer to the natural language question.

Brief Summary Text (31):
According to an aspect of the invention, the NLIR module defines an interface including a first interface method for receiving the text documents from the search engine and returning the hash-table fingerprints to the search engine. The interface defined by the NLIR module also includes a second interface method for receiving a current natural language question and a hash-table fingerprint for a current document from the search engine, and returning a document score to the search engine connoting the likelihood that the current document contains an answer to the natural language question. The interface defined by the NLIR module may also include a third interface method for initiating processing of the natural language question, and a fourth interface method for terminating processing of the natural language question.

Drawing Description Text (4):
FIG. 3 is a functional block diagram that illustrates a document including a hash-table fingerprint that is created and searched by the natural language information retrieval utility.

Detailed Description Text (7):
The preexisting NLP core code module (i.e., the base technology) has a number of shortcomings including (1) the set of LFTs for a document is very large, and storing the LFTs for a large document set requires a large memory allocation; (2) literal LFT matching for a large document set is very time consuming; (3) the base technology is not "thread safe" and, thus, does not run efficiently on multi-processor platforms; and (4) LFTs represented as "fingernails" stored as hash values cannot be directly identified, which makes LFT generation and scoring difficult to analyze after the representations of the LFTs have been stored in an associated fingernail.

Detailed Description Text (8):
The present invention solves these problems through a number of techniques. The memory storage and literal LFT matching problems are solved by storing and searching hash tables that represent the LFTs rather than the LFTs themselves. That is, each document is "indexed," which means that it is represented by a hash-table fingerprint that corresponds to the LFTs identified by the base technology for the document. The hash table is populated by using a Cyclical Redundancy Check (CRC) algorithm to compute a 32-bit CRC quasi-random hash value corresponding to the literal string forming each LFT. For example, the CRC defined by ISO 3390, which is well known to those skilled in the art, may be used to compute the hash values. The upper 16 bits of the CRC value are used to determine an "address hash" or array index number for the hash table, and the lower 19 bits are used as a "signature hash" that is stored within the array entry corresponding to the array index (the upper three bits of the signature hash overlap with the lower three bits of the address hash). This hash-table technique is particularly well suited to the natural language information retrieval application because an occasional hash-related mistake or "collision" is not catastrophic; it just results in a document having a higher score than it otherwise would have.

Detailed Description Text (9):
The number of elements in the hash table is equal to 110% times the number of LFTs in the document to provide "padding" in the table. The hash table values A(i) are initialized to zero. An array entry point (array index=i) for a particular LFT is computed as "i=hash mod (N)," which produces an address (i) between zero and N-1. Specifically, the array entry point (i) is set equal to the remainder of address hash/N. If the array entry A(i) for that address is not equal to zero (i.e., the table entry A(i) corresponding to address (i) is already occupied by a previously-assigned signature hash), then the array index is incremented. If the resulting array index is outside the array (i.e., array index=N), then the array index is set equal to zero (i.e., the address value wraps from the bottom to the top of the array). Once an array index with an empty array entry (i.e., A(i)=0) is located, the signature hash for the LFT is stored in that array entry. This process is repeated until the signature hash values for all of the LFTs are stored in the hash table.

Detailed Description Text (10):
Those skilled in the art will appreciate that incrementing the array index is a simple method for identifying additional candidate locations to store the signature hash. Other more sophisticated methods could also be used, such as adding a quasi-random number to the array entry point. For example, the quasi-random number could be based on the LFT and the number of candidate locations already considered.

Detailed Description Text (12):
During question processing, each document in the universe of indexed documents is searched using a method that analogous to the method used to store the LFT signatures. To illustrate question processing, consider the example in which alternative array candidates are identified by incrementing the array index. The base technology first computes LFTs for a natural language question. A 32-bit CRC is then computed using the same CRC algorithm that was used during index processing. The upper 16 bits of the CRC are used to determine an array index for an array entry point (i). The array entries for that array index (i) and successive non-empty array entries are checked in the fingerprint (i.e., hash table) for a particular document. If an array entry A(i) is found matching the lower 19 bits of the CRC (i.e., the signature hash for the LFT), this is considered a match for the particular LFT. If an empty data entry (i.e., A(i)=zero) is found before a match, this is considered a lack of a match for the particular LFT.

Detailed Description Text (13):
Note that the 110% "padding" limits the amount of the hash table that must be searched for each question LFT. This question-LFT matching process is repeated for each LFT in the natural language question, and the scores for the resulting LFT matches are summed to produce a document score. This process is also repeated for one or more document in the universe of indexed documents. The documents are then ranked in the order of document score and presented to the user as documents that have a high likelihood of containing an answer to the natural language question.

Detailed Description Text (14):
Representing LFTs using pseudo-random numbers stored in a hash-table will inevitably result in a certain number of "collisions" in which two different LFTs produce the same hash value. Collisions are inevitable, of course, because the number of possible 19-bit signature hash values is less than the number of possible LFTs that occur in the English language. Using the hash-table technique reduces but does not eliminate the probability of a collision. Nevertheless, the hash-table technique is advantageous in the context of an NLIR search engine because, in this context, the consequences associated with a collision are relatively minor. In particular, the only consequence associated with a collision will typically be that a particular document will receive a higher score than it would have received in the absence of the collision.

Detailed Description Text (15):
The exemplary embodiments of the present invention recognize that this type of occasional over-score is quite acceptable in the context of an NLIR system that presents a user with a ranked list of potentially-relevant documents in response to a natural language question. The occasional over-score is quite acceptable because the user can easily disregard an over-scored document if it is, in fact, not relevant to the user's inquiry. Other documents in the ranked list will, most likely, not be over-scored. Moreover, the alternatives to using a hash-table technique, such as storing a complete LFT listing for each document in the universe of searchable documents, computing LFTs on the fly for each document in the universe of searchable documents, or foregoing NLIR processing are far less attractive.

Detailed Description Text (18):
Those skilled in the art will appreciate that the specific parameters selected for the exemplary embodiment, such as the 110% padding factor used to determine the size of the hash-table array, the 32-bit size of the hash value, the 16-bit size of the address hash, and the 19-bit size of the signature hash may all be varied somewhat within the teachings of the present invention. Accordingly, the number of LFTs that may be stored in a hash-table fingerprint for a particular document may be increased or decreased by altering the number of bits in the address hash. And the likelihood of LFT "collisions" caused by different LFT producing matching hash values can be increased or decreased by altering the number of bits in the hash value.

Detailed Description Text (19):
More specifically, the particular values selected for these parameters represent trade-off balances struck between the competing goals of reducing memory requirements, increasing processing speed, and increasing searching precision. These trade-off balances may be altered somewhat in alternative embodiments of the invention, particularly in view of the trend of increasing processing speed and memory-storage capabilities prevailing in computer technology. In addition, the specific LFTs identified by the NLP core code module and the heuristic scores assigns to LFT matches may also be varied somewhat within the teaching of the present invention. Alternate embodiments of the invention may also employ techniques other than the CRC algorithm defined by ISO 3309 for computing pseudo-random numbers used as hash values, and may use logical-form relationships other than LFTs, such as logical-form relationships involving three, four, or more words in semantic constructs, Boolean logical expressions, and so forth.

Detailed Description Text (31):
During index processing, the search engine 104 passes a text-containing document 106 to the NLIR utility 101, which returns a hash-table fingerprint 108 to the search engine 104. The hash-table fingerprint 108, which is opaque to the search engine 104, contains a highly compressed representation of LFTs contained within the document 106. The search engine 104 may pass additional documents to the NLIR utility 101 for index processing to create and add to the group of searchable documents 102, which is represented by documents 106a-n having associated hash-table fingerprints 108a-n. Thus, the search engine 104 selects documents for index processing, and the NLIR utility 101 provides the search engine with a tool for making the selected documents amenable to NLIR processing.

Detailed Description Text (32):
More specifically, the search engine 104 passes a representative text-containing document 106 to an NLIR module 110, which cooperates with a proxy process module 112

and an NLP core code module 114 to create the corresponding hash-table fingerprint 108. The NLP core code module 114 relies heavily on the use of global variables and, for this reason, cannot run multiple threads simultaneously. Therefore, if multiple LFT queries were configured as multiple threads, the NLP core code module 114 would not be able to run multiple LFT queries on multiple processing units 21a-n simultaneously. This limitation would undermine much of the advantage of running the NLIR system 100 on the multi-processor computer system 20.

Detailed Description Text (35):
The NLIR module 110 obtains LFTs for each sentence of the representative document 106 in the manner described above. The NLIR module 110 then engages in hash operations 122 to create the hash-table fingerprint 108, which represents each LFT as a pseudo-random number. Specifically, the NLIR module 110 allocates a 19-bit array having a size "N" that is equal to 110% times the number "M" of LFTs for the document. The NLIR module 110 then populates the array using the ISO 3309 CRC algorithm to compute a 32-bit hash value corresponding to the literal string forming each LFT. The upper 16 bits of each hash value are used to determine an array entry point or array index, and the lower 19 bits are used as a "signature hash" that is stored within the array. If the array entry corresponding to the array entry point is not empty (i.e., contains a previously-assigned signature hash), the NLIR module 110 increments the array index until an empty array entry is located. The NLIR module 110 then stores the signature hash for the LFT in that array entry.

Detailed Description Text (39):
While the NLIR module 110 maintains an LFT list 120 for a particular question, the search engine 104 may pass an LFT comparison requests 125 to the NLIR module. Each LFT comparison request 125 includes two "handles" that specify a current document and a current natural language question for LFT comparison. For each question LFT, the NLIR module 110 determines whether the current document contains a matching hash value. Specifically, the NLIR module 110 computes a hash value for the question LFT using the ISO 3309 CRC algorithm and uses the upper 16 bits of the hash value as an index hash and the lower 19 bits of the hash value as a signature hash. The NLIR module 110 then determines whether the hash-table fingerprint for the current document includes the signature hash at an array index corresponding to the index hash. The NLIR module 110 follows this procedure to identify zero or more matches between the question LFTs and the hash-table fingerprint for the current document.

Detailed Description Text (42):
FIG. 3 is a functional block diagram that illustrates a document including a hash-table fingerprint 300 that is created and searched by NLIR utility 101. The hash-table fingerprint 300 is typically an array of 19-bit values A(i) in which each value corresponds to a 16-bit array index (i). The hash-table fingerprint 300 includes "N" array entries, where "N" is equal to 110% times the number "M" of LFTs in the corresponding document. The hash-table fingerprint 300 stores representations of 32-bit hash values that may be computed using the ISO 3309 CRC algorithm. Specifically, the array index (i) corresponds to the address hash 302, which is the upper 16 bits of a hash value. The value stored within an array element correspond to a signature hash value, which in the lower 19 bits of the hash value.

Detailed Description Text (43):
As the address hash 302 is a 16-bit value, the maximum size of the hash-table fingerprint 300 is 65,536, which corresponds to a maximum number of LFTs for a document of approximately 59,578. The signature hash value, which is a 19-bit number, permits up to 524,288 different signature hash values. In the rare case in which 110% times the number of LFTs in a document exceeds 65,536, the entire 32-bit CRC is sorted and stored in a 32-bit array during index processing. This array is searched using a binary searching technique on an LFT-by-LFT basis during question processing.

Detailed Description Text (44):
FIG. 4A is a functional block diagram that illustrates an NLIR.DLL interface 400 for the NLIR module 110. The NLIR.DLL interface 400 includes an NLIR_ParseDocument interface method 402 that the search engine 104 calls to obtain a hash-table fingerprint for a document. The NLIR_ParseDocument interface method 402 returns the hash-table fingerprint, which is opaque to the search engine 104. Because the LFTs are represented by opaque entries in a hash table, the LFTs as represented in the hash

table cannot be viewed directly. The debugging module 130 allows a user activate and deactivate trace functions that cause the NLIR module 110 to generate the actual LFTs for analysis. The debugging module 130 is described in greater detail below with reference to FIG. 4B.

Detailed Description Text (54):
Step 516 and the "NO" branch from step 514 are followed by step 518, in which the NLIR module 110 determines whether the document contains another sentence. If the document does include another sentence, the "YES" branch loops from step 518 to step 504, in which the NLIR module 110 parses another sentence from the document. If the document does not include another sentence, the "NO" branch is followed from step 518 to step 520, in which the NLIR module 110 determines the number of LFTs "M" for the document. Step 520 is followed by step 522, in which the NLIR module 110 allocates an array having "N" 19-bit entries, where "N" is equal to "M" times 110%. Step 522 is followed by routine 524, in which the NLIR module 110 creates a hash-table fingerprint for the document by assigning the LFTs for the document to the array. Following routine 524, the document is a member of the group of searchable documents 102 that may be accessed by the NLIR utility 101 during subsequent question processing. Routine 524 is described in greater detail with reference to FIG. 6.

Detailed Description Text (56):
FIG. 6 is a logic flow diagram that illustrates routine 524, in which the NLIR module 110 assigns "M" LFTs for a current document, which were identified by the NLP core code module 114, to the array of size "N" (N=M.times.110%) to create a hash-table fingerprint for the current document. Routine 524 begins following step 522, shown in FIG. 5. In step 602, the NLIR module 110 initializes the elements of the array (i.e., sets A[i]=0 for i=0 through N-1). Step 602 is followed by step 604, in which the NLIR module 110 gets one of the LFTs for the current document in a text string format. Step 604 is followed by step 606, in which the NLIR module 110 computes a hash value for the LFT, typically by applying the CRC algorithm defined by ISO 3309 to the LFT text string. In other words, the NLIR module 110 computes a 32-bit hash value, which is a pseudo-random number corresponding to the LFT text string.

Detailed Description Text (57):
Step 606 is followed by step 608, in which the NLIR module 110 parses the hash value by setting a signature hash for the LFT to the lower 19 bits of the hash value. Step 608 is followed by step 610, in which the NLIR module 110 sets an address hash for the LFT to the upper 16 bits of the hash value. Step 610 is followed by step 612, in which the NLIR module 110 computes an array entry point for the LFT based on the address hash. Specifically, the array entry point may be computed as the remainder of the number of elements in the array "N" divided by the address hash (i.e., array entry point=address hash mod (n)). The purpose of this calculation is to convert the 16-bit address hash into a pseudo-random number having a value between zero and N-1, which causes the array entry point (array index=i) to correspond to the index value for one of the array elements.

Detailed Description Text (58):
Step 612 is followed by step 614, in which the NLIR module 110 determines whether the value (A[i]) stored at the array entry point (array index=i) is equal to zero, indicating that a hash value has not yet been stored at that particular array element. If the value (A[i]) stored at the array entry point (array index=i) is equal to zero, indicating that a hash value has not yet been stored at that particular array element, the "YES" branch jumps to step 622, which is described below. On the other hand, if the value (A[i]) stored at the array entry point (array index=i) is not equal to zero, indicating that a hash value has already been stored at that particular array element, the NLIR module 110 increments the array index (i.e., array index=i+1). Step 616 is followed by step 618, in which the NLIR module 110 determines whether the newly-computed array index is larger than the largest index value in the array (i.e., array index=N).

Detailed Description Text (59):
If the newly-computed array index is larger than the largest index value in the array (i.e., i=N), the "YES" branch is followed from step 618 to step 620, in which the NLIR module 110 sets the array index to zero (i.e., the array index loops from the bottom to the top of the array). From step 620 and the "NO" branch from step 618, routine 518

loops to step 614, in which the NLIR module 110 checks whether the value stored at the new array index is equal to zero. Because the number "N" of elements in the array is larger than the number "M" of LFTs for the current document, the NLIR module 110 will eventually loop through the steps 614 through 620 until it locates an empty (i.e., A[i]=0) array element. Once the NLIR module 110 identifies an empty array element, the "YES" branch jumps from step 614 to step 622, in which the NLIR module 110 stores the signature hash for the current LFT in the empty array element.

Detailed Description Text (60):
Step 622 is followed by step 624, in which the NLIR module 110 determines whether there is another LFT for the current document to assign to the array. If there is another LFT for the current document, the "YES" branch loops from step 624 to step 604, in which the NLIR module 110 gets another LFT. If there is not another LFT for the current document, the "NO" branch is followed from step 624 to the "END" step 626, which returns to step 526 shown on FIG. 5. Routine 518 thus allows the NLIR module 110 to assign each LFT for the current document to the array to create a hash-table fingerprint for the current document.

Detailed Description Text (64):
Step 712 is followed by step 714, in which the NLP core code module 114 returns LFTs for the question to the calling thread (i.e., to the NLIR module 110). Once the NLIR module 110 has obtained the LFTs for the question, it is ready to compare these question LFI's to the document LFTs represented by the hash-table fingerprints 108a-n for the documents in the group of searchable 102. Thus, step 714 is followed by step 720, in which the search engine 104 passes a comparison command to the NLIR module 110, typically by calling the NLIR_CheckDocQuery interface method 406. The search engine 104 specifies a particular document and a particular question to compare when calling the NLIR_CheckDocQuery interface method 406. Step 720 is followed by routine 722, in which the NLIR module 110 compares the LFTs for the question to the LFTs for the specified document. Also in routine 722, the NLIR module 110 computes a document score based on the comparison and returns the document score to the search engine 104. Routine 722 is described in greater detail with reference to FIG. 8.

Detailed Description Text (67):
FIG. 8 is a logic flow diagram that illustrates routine 722, in which the NLIR module 110 computes a document score for the natural language question. Routine 722 begins following step 720 shown on FIG. 7. In step 802, the NLIR module 110 initializes (i.e., sets to zero) a document score for the current document. Step 802 is followed by step 804, in which the NLIR module 110 gets one of the document LFTs as a text string. Step 804 is followed by step 806, in which the NLIR module 110 computes a 32-bit hash value for the LFT using the same algorithm that was used to create the hash-table fingerprints 108a-n for the documents in the group of searchable documents 102. For example, the CRC routine defined by ISO 3309 may be used for both purposes.

Detailed Description Text (68):
Step 806 is followed by step 808, in which the NLIR module 110 parses the lower 19 bits of the hash value as a signature hash for the LFT. Step 808 is followed by step 810, in which the NLIR module 110 sets an address hash for the LFT to the upper 16 bits of the hash value. Step 810 is followed by step 812, in which the NLIR module 110 computes an array entry point for the LFT based on the address hash. Specifically, the array entry point may be computed as the remainder of the number of elements in array "N" divided by the address hash (i.e., array entry point=address hash mod (n)). The purpose of this calculation is to convert the 16-bit address hash into a pseudo-random number having a value between zero and N-1, which causes the array entry point (array index=i) to correspond to the index value for one of the array elements. It should be noted that the procedure described above for steps 804-812 followed during question processing is identical to the procedure described in steps 604-612 followed during index processing.

Detailed Description Text (69):
Step 812 is followed by step 814, in which the NLIR module 110 compares the signature hash for the question LFT to the entry (A[i]) stored at the array entry point. Step 814 is followed by step 816, in which the NLIR module 110 determines whether there is an LFT match at the current array index (i), which is initially set to the array entry point. That is, in step 816, the NLIR module 110 determines whether the signature hash

for the question LFT is the same as the entry (A[i]) stored at the current array index (i). If there is an LFT match at the current array index, the "YES" branch is followed to step 818, in which the NLIR module 110 looks up an LFT score for the current LFT and adds this LFT score to the document score for the current document. For example, the NLIR module 110 may look up one of the LFT scores shown in Table 1, above, based on the type of matching LFT.

Detailed Description Text (71):
Referring again to step 818, if a matching LFT is identified, routine 716 jumps to step 828 after the LFT score has been added to the document score. And referring again to step 820, if an empty array entry is encountered indicating that the current document does not include a match for the current LFI, the "YES" branch jumps from step 820 to step 828, which ends the processing for the current LFT. Thus, the NLIR module 110 identifies the current LFT as a matching LFT if the current signature hash matches the data value stored at any of the consecutively-addressed data-containing memory locations beginning with the array entry point.

Current US Original Classification (1):
707/5

Current US Cross Reference Classification (1):
707/100

Current US Cross Reference Classification (2):
707/102

Current US Cross Reference Classification (3):
707/3

CLAIMS:

1. A method for creating a group of searchable documents comprising the steps of, for each of a plurality of documents: receiving text defining the document; parsing the text into a plurality of text portions; obtaining one or more logical form relationships corresponding to each text portion; defining an array having a size corresponding to the number of logical form relationships for the document; and creating a hash-table fingerprint for the document by, for each logical form relationship, computing a hash value, obtaining an address hash and a signature hash based on the corresponding hash value, and storing the signature hash in the array at a memory location corresponding to the address hash.

2. The method of claim 1, further comprising the steps of: receiving a natural language question; obtaining one or more logical form relationships for the question; computing a hash value corresponding to each logical form relationship for the question; and for each document in the group of searchable documents, comparing each hash value corresponding to the logical form relationships for the question to the hash-table fingerprint for the document and identifying one or more matching hash values, obtaining a score for each matching hash value, and computing a document score connoting the likelihood that the document contains an answer to the question by summing the score for each matching hash value.

3. The method of claim 2, further comprising the steps of, for a current hash value for the question: parsing the current hash value into a current address hash and a current signature hash; identifying an array entry point in the array corresponding to the current address hash; and if the array entry point is not empty, identifying one or more consecutively-addressed data-containing memory locations beginning with the array entry point, comparing the current signature hash to the data value stored at each of the consecutively-addressed data-containing memory locations, and if the current signature hash matches the data value stored at any of the consecutively-addressed data-containing memory locations, identifying the current hash value as a matching hash value.

4. The method of claim 3, further comprising the steps of, for a current hash value for the question: if the array entry point is empty, identifying the current hash value as a non-matching hash value; and if the current signature hash does not match

the data value stored at any of the consecutively-addressed data-containing memory locations, identifying the current hash value as a non-matching hash value.

6. A computer-readable medium having computer-executable instructions comprising: a natural language information retrieval module configured for: creating a group of searchable documents by, for each document, receiving text defining the document from a search engine and returning a hash-table fingerprint comprising a representation of logical form relationships for the document to the search engine, and for each document, receiving a natural language question and the hash-table fingerprint comprising the representation of logical form relationships for the document from the search engine and returning a document score to the search engine connoting the likelihood that the document contains an answer to the natural language question; and the search engine configured for: ranking the documents in order of their respective document scores, and displaying a list of highest ranking documents as a suggestions list of documents containing an answer to the natural language question; and wherein the natural language information retrieval module is configured for parsing each document into a plurality of sentences, further comprising a proxy process module configure for: receiving the sentences from one or more active client threads other than the first active client thread, each active client thread associated with the natural language information retrieval module; creating a process for each client thread other than the first active client thread; and passing the sentences for each client thread other than the first active client thread to a natural language processing core code module in the context of an associated process, the natural language processing core code module configured to identify one or more logical form relationships corresponding to each sentence and return the logical form relationships to the natural language information retrieval module.

7. A computer-readable medium having computer-executable instructions comprising: a natural language information retrieval module configured for: creating a group of searchable documents by, for each document, receiving test defining the document from a search engine and returning a hash-table fingerprint comprising a representation of logical form relationships for the document to the search engine, and for each document, receiving a natural language question and the hash-table fingerprint comprising the representation of logical form relationships for the document from the search engine and returning a document score to the search engine connoting the likelihood that the document contains an answer to the natural language question; and the search engine configured for: ranking the documents in order of their respective document scores, and displaying a list of highest ranking documents as a suggestions list of documents containing an answer to the natural language question; and further comprising a debugging module defining an interface comprising: a first interface method for activating and deactivating a trace document function that, when active, causes the natural language information retrieval module to identify the logical form relationships identified for document text processed by the natural language information retrieval module; and a second interface method for activating and deactivating a trace question function that, when active, causes the natural language information retrieval module to identify the logical form relationships identified for questions processed by the natural language information retrieval module.

L27: Entry 7 of 8 File: USPT Jun 13, 2000

DOCUMENT-IDENTIFIER: US 6076084 A
TITLE: File transfer method and apparatus utilizing delimiters

Abstract Text (1):
The present invention facilitates the transmission of a file to a computer where the receiving computer has a file (called the old file) that is related to the file being transmitted (called the new file) but where the sending computer does not know the status or content of the old file. As a preliminary step, one of the computers generates a Delimiter Selection Profile Table (DSPT). The DSPT is generated by first determining the number of times each delimiter in a set of delimiters appears in the file and the distance between the locations of the delimiters in the file. Next using the information in the DSPT one of the delimiters is chosen as the delimiter which will be used and this delimiter is transmitted to the computer which did not generate the DSPT. The receiving computer next generates a Segment Profile (SPT) of the old file and the sending computer generates an SPT the new file. The SPT is generated by calculating a hash code (such as a CRC) for each segment which is defined by the selected delimiter. The hash codes from the old file are transmitted to the sending computer. The sending computer then sends to the receiving computers those segments in the new file that do not have a hash code number which matches one of the hash code numbers from the old file. The sending computer also sends an indication of where segments from the old file should be inserted into the new file. The receiving computer then constructs the new file from the segments received from the appropriate old segments.

Brief Summary Text (9):
The present invention facilitates the transmission of a file to a computer where the receiving computer has a file (called the old file) that is related to the file being transmitted (called the new file) but where the sending computer does not know the status or content of the old file. With the present invention as a preliminary step, one of the computers generates a Delimiter Selection Profile Table (DSPT). Either the receiving computer generates a DSPT of the old file or the sending computer generates a DSPT of the new file. The DSPT is generated by first determining the number of times each delimiter in a set of delimiters appears in the file and the distance between the locations of the delimiters in the file. Next using the information in the DSPT one of the delimiters is chosen as the delimiter which will be used and this delimiter is transmitted to the computer which did not generate the DSPT. The receiving computer next generates a Segment Profile (SPT) of the old file and the sending computer generates an SPT the new file. The SPT is generated by calculating a hash code (such as a CRC) for each segment which is defined by the selected delimiter. The hash codes from the old file are transmitted to the sending computer. The sending computer then sends to the receiving computers those segments in the new file that do not have a hash code number which matches one of the hash code numbers from the old file. The sending computer also sends an indication of where segments from the old file should be inserted into the new file. The receiving computer then constructs the new file from the segments received from the appropriate old segments.

Detailed Description Text (9):
d) Both the new and the old file are divided into segments based on the selected delimiter and the segments are analyzed and a Segment Profile Table (SPT) is generated for both the old and the new files. A hash number for each segment is generated.

Detailed Description Text (10):
e) Those segments in the new file which do not have a hash number corresponding to the

hash number of a segment in the old file are sent to the receiving computer, and

Detailed Description Text (11):
f) The receiving computer combines the parts of the new file that were transmitted with segments from the old file which have hash numbers corresponding to segments in the new file to construct a replica or copy of the new file at the receiving computer.

Detailed Description Text (32):
The segment profile table gives three numbers for each segment. The first number is an offset which tells where the segment begins and the second number is the length of the segment. The third number in the SPT relative to each segment is a hash number or CRC for the segment. It is noted that the hash number used herein is the CRC (cyclical redundancy check number) however, it can be anyone of the well known types of number hash numbers which generate a unique identifying number from a series of bits or bytes. It is also noted that while a CRC or other hash number is described herein as defining a unique segment, mathematically duplication or errors are possible. The likely hood of such errors is so remote (less likely than a failure in the typical computer hardware) that they can for the purposes of the present invention be considered to be unique. It is also noted that to decrease the likelihood of error and to facilitate computation, two concatenated CRC number can be used. The particular technique used to generate hash numbers (e.g. CRC numbers) forms no part of the present invention and can be conventional. The algorithm in FIG. 6 is used to calculate the entries in the SPT for each of the files. The sending computer calculates these values for the new file and the receiving computer calculates these numbers for the old file.

Current US Original Classification (1):
707/1

Current US Cross Reference Classification (1):
707/2

WEST Search History

DATE: Thursday, June 12, 2003

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=NO; OP=ADJ			
L33	l21 and (l29 or l16)	0	L33
L32	l29 and l17	0	L32
L31	l29 and (l28 or l21 or l16 or l8 or l7 or l6 or l4)	1	L31
L30	L29 and l1	7	L30
L29	(hash$3 and algorithm$1).ab.	77	L29
L28	"cyclical redundancy check algorithm"	7	L28
L27	L26 and l9	8	L27
L26	(l4 or l16) and hash$3	34	L26
L25	L24 and l9	0	L25
L24	l16 and (l6 or l7 or l8)	26	L24
L23	l4 and (l6 or l7 or l8)	0	L23
L22	L21 and l1	5	L22
L21	key same record$1 same index$3 same catalog$3	23	L21
L20	L19 and (l6 or l7 or l8 or l16)	0	L20
L19	l1 and l2 and l3	4	L19
L18	L17 and l9	3	L18
L17	L16 and (l2 or l3 or l5 or l6 or l7 or l8)	41	L17
L16	"cyclical redundancy check"	712	L16
L15	l4 and l9	1	L15
L14	l4 and l1	0	L14
L13	(l2 or l3) and l4 and (l6 or l7 or l8 or l5)	0	L13
L12	l3 and l4 and l5 and (l6 or l7 or l8) and l9	0	L12
L11	l2 and l3 and l4 and l5 and (l6 or l7 or l8) and l9	0	L11
L10	l2 and l3 and l4 and l5 and (l6 or l7 or l8) and l1	0	L10
L9	((707/$)!.CCLS.)	10842	L9
L8	CRC-32	137	L8
L7	CRC-16	249	L7
L6	CRC-CCITT	80	L6
L5	index same key	6452	L5
L4	"cyclical redundancy check value"	21	L4
L3	record adj1 address	742	L3
L2	determining near3 key	2741	L2
L1	(707/1 OR 707/3).CCLS.	3084	L1

END OF SEARCH HISTORY

WEST

Generate Collection | Print

L35: Entry 2 of 12 File: USPT Feb 25, 2003

DOCUMENT-IDENTIFIER: US 6526418 B1
TITLE: Systems and methods for backing up data files

Brief Summary Text (13):
The synchronization replication process may include an image capture mechanism that can process a file system or data structure on the server and create a file system image signal that is representative of the state, or a state, of the file system at a particular time. For example, the image capture mechanism can include a directory processor that can process a directory structure such as a conventional UNIX file system or windows NT file system directory structure, to create a directory image signal which is representative of a state of the directory at the time of processing that directory structure. In one embodiment the image generator operates by processing metadata associated with a data structure of file structure such as the size of the data structure, the size of directory, the name to the files and directory, the metadata associated with the last time the directory was edited, or when it was created, the n umber of files in the directory, and other such metadata. A hashing process of cyclical redundancy check (CRC) process may be applied to the metadata to come up with an identifier that is uniquely, or substantially uniquely, representative of the state of the processed file structure at the time of the processing. A similar image generator process may be employed for processing a file system on the server to create a file system image signal that is representative of a state of a file, directory, or the data within a file. In either case, the image signal is representative of a state of the file structure, such as the directory, a group of files in a directory, a particular file, or even a particular portion of a file.

Detailed Description Text (26):
After step 98 the process 90 proceeds to step 100 wherein the meta data for the 32 directories is processed to generate an image signal representative of a state of the 32 directories. To process the meta data, the backup system may apply a hashing algorithm, a cyclical redundancy check algorithm, or any other suitable process for generating an image signal that may be representative of the state of the 32 directories.

WEST

End of Result Set

Generate Collection | Print

L15: Entry 1 of 1 File: USPT Aug 13, 2002

DOCUMENT-IDENTIFIER: US 6434558 B1
TITLE: Data lineage data type

Current US Original Classification (1):
707/6

Current US Cross Reference Classification (1):
707/101

CLAIMS:

17. The data structure as recited in claim 16 wherein the compressed GUID is based on a cyclical redundancy check value derived from a sixteen-byte GUID.

L18: Entry 2 of 3 File: USPT Feb 9, 1993

DOCUMENT-IDENTIFIER: US 5185857 A
TITLE: Method and apparatus for multi-optional processing, storing, transmitting and retrieving graphical and tabular data in a mobile transportation distributable and/or networkable communications and/or data processing system

Detailed Description Text (17):
Upon receipt, the data is, if compressed, decompressed, parsed by the on-board host 41, and checked for errors 40. The present invention incorporates a cyclical redundancy check (CRC) to determine if there were errors during transmission. If there were errors during transmission, an error status signal is generated 40 and the packet is returned to the seat computer via transmission lines 39. If there were no errors during transmission, the host control program determines the Application ID 42 and the configuration of the seat computer 44 which transmitted the data. This enables the on-board host to format the ultimate command so as to be compatible with the seat mounted system. In operation (e.g. transmit) the host computer 14 can utilize any suitable software such as SCO Xenix or Open Desktop software or any hardware and/or software package providing necessary options and commands for all end users on the mobile system. The Application ID distinguishes the particular device, site and application which will use the data sent in the packet (e.g. GLS a graphic-listing service for real estate, architectural designs or medical diagnostics and/or imaging duties).

Detailed Description Text (23):
PACKET DATA--contains actual data (hardware and graphics configuration information for end user workstation, encryption key, number of records matching query, index for GRDB and (up to) four database records selected by end user database query) being sent in the Packet. These files may or may not be present depending on the Record Type.

Current US Cross Reference Classification (1):
707/104.1

Current US Cross Reference Classification (2):
707/4

L18: Entry 1 of 3 File: USPT Apr 5, 1994

DOCUMENT-IDENTIFIER: US 5301350 A
TITLE: Real time storage/retrieval subsystem for document processing in banking operations

Detailed Description Text (80):
Optical Transmitter Interface: The Optical Transmitter Interface 40 (FIG. 3B) of the Optical Link Controller 10.sub.po transmits serial data from the Imaging Module 8.sub.1 over the fiber optic link where a fiber optic transmitter converts electrical signals to optical signals. The Optical Transmitter 40 receives parallel data from storage, serializes the data, encodes the data using Manchester encoding, and then transmits the serial data over the fiber optic link 9.sub.po. Internal control logic handles the encoding and transmission of data, which logic also generates a cyclical redundancy check to ensure that the transmitter data can be verified at the receipt point. Additionally, the logic frames the data being transmitted.

Detailed Description Text (199):
A "record" must have one key field. The key field houses a supplemental data structure called an "index".

Detailed Description Text (205):
The file system name is the name of the file system that the particular file is located in. This is a disk organization parameter configured into the system. The file name is the name of the file the image or images are stored into. This is the "structured file" type. There can be many files defined or created within a file system. The retrieval index (RIX) is the identifier of the specific record within a file, and is used for retrieval purposes. This is sometimes referred to as the record "key". The financial information system (FIS) "system directory", which resides in the host 6, is configured with several authorizations, some of which define and specify which users or programs can access images. This information is loaded into a storage/retrieval module for verification when needed. Generally, the information required for retrieval would follow sequences such as: FILE SYSTEM; FILE NAME; RIX; PROGRAM NAME.

Detailed Description Text (217):
In those fields containing program-supplied data, one "key" field is required. Each key, which consists of all the data within a field, gives rise to a supplemental structure called an "index". The existence of an index for a key allows searches for records with particular key values to be performed in an optimized fashion. Indexes are automatically maintained by the system in an index file. The index file is created by the file management system when the structured file is created.

Detailed Description Text (222):
Retrievals from structured files are performed in terms of "retrieval sequences". A retrieval sequence is a subset of the records within a structured file and might be either a range of key values within an index file or a sequence of random key values. After a retrieval sequence has been specified for an association, services can be invoked to retrieve the individual records/fields that make up the sequence.

Detailed Description Text (241):
Index File Structures: An Index File contains Index Records which is a structure providing information as to the record index, size, the number of fields, the record index number, the field directory entry for a series of fields from a first field through the nth field. Then each index record structure can be organized into a graph

of index structures which start from a low value and continue on to a high value for the nth index number. This allows data records to be located by means of a key value which is normally a part of the data record itself.

Detailed Description Text (243):
If all of the records in a data file are of variable length, a special kind of Index File is used to locate an arbitrary record by its record number. This type of Index File is given the term "Record Index File" and it contains index records with no Key value. In this case the record "n" is located in the Data File by first locating the nth Index Record in the record index file and then using the contents of this index record to locate the data record.

Detailed Description Text (386):
RIX: This is the retrieval index which is a unique key used to retrieve any stored image. Elements of the "key" include the data, the location of capture, the sorter identification, and the sequence of input.

Current US Cross Reference Classification (1):
707/104.1

Current US Cross Reference Classification (2):
707/200

CLAIMS:

1. In a bank check document handling system for capturing image and information data of negotiated bank checks for bank record processing and which system is managed by a host computer, a storage/retrieval module subsystem for storing said image and information data for retrieval and conveyance to any one of a plurality of image work stations and printers for conversion to human readable format, said storage/retrieval module subsystem comprising:

(a) means for receiving digitized optical signals containing bank check document image packets having (i) image data and (ii) sequential non-image information data related to said image data;

(b) means for converting said digitized optical signals to digitized electrical signals forming said bank check document image packets;

(c) storage operation means for storing, in real time, said bank check document image packets on identified areas of magnetic disk units via a file management system which includes:

(c1) first storage file means for storing non-image bank check document data as a plurality of sequential files made up of a sequence of bytes of data;

(c2) second storage file means for storing, in real time, said bank check document image packets in a structured file system made of a plurality of records where each record has a key field with an index identifying each record;

(d) means for retrieving, in real time, a selected bank check document image packet while simultaneous and concurrent storing operations of bank check document image packets are taking place, said means for retrieving including:

(d1) means for selecting said first storage file means or said second storage file means to effectuate retrieval of either or both non-image information and/or a selected bank check document image packet for transmission to a requesting work station or printer;

(e) means for transmitting retrieved bank check document image packets to a work station or printer for display;

(f) means for communicating with a host computer to receive operational instructions and to transmit retrieved sequential non-image information for use by said host computer.

16. In a bank check handling system, controlled by a main host computer, for capturing images and non-image item data from digitized bank check document image data packets being received, a storage and retrieval subsystem for storing compressed image data of the front and back of each bank check document received, and for storing related non-image item data such as customer account number and dollar value amount for each said received bank check document, the storage/retrieval subsystem comprising:

(a) a minimum of two disk drive means which include first and second disk drive units for operating first and second disk storage units;

(b) said first and second disk storage units being organized into identified files in a file management system;

(c) first and second disk controller means for controlling said disk drive means;

(d) a first storage processor means for receiving said digitized bank check document image data packets for storage on said first and second disk storage units in real time via said first and second disk controller means and said first and second disk drive means, said first storage processor means including:

(d1) first buffer memory means for temporarily storing at least two of said bank check document image data packets;

(d2) a file management system for;

(i) filing said related non-image item data in a first sequentially ordered file of consecutive bytes, and for

(ii) filing said bank check document image packet in a second structured file made of a plurality or records in which each record has a key field with an index identifying each record,

(e) a second unit processor means for retrieving, in real time, selected non-image item data or said bank check document image data packets via said first and second disk controller means and said first and second disk drive means for transmission to a requesting work station or printer, said second unit processor means including:

(e1) second buffer memory means for temporarily storing at least two bank check document image data packets;

(e2) means for executing retrieval operations simultaneously and concurrently with the execution of storage operations by said first storage processor means;

(f) optical link controller means for receiving digitized optical signal data in digitized packets and converting said optical signal data to digitized electrical signals for transmission on a parallel system bus means to said first storage processor means;

(g) first local area network controller means for communication between a plurality of operator work stations and for facilitating data requests to said second processor means for retrieval, and for transmitting requested image and non-image item data to a requesting work station;

(h) second local area network controller means for communicating with said main host computer;

(i) said parallel system bus means for enabling concurrent data exchange between said first and second processor means, said first and second disk controller means, said optical link controller means, and said first and second local area network controller means.

L22: Entry 2 of 5 File: USPT Aug 8, 1995

DOCUMENT-IDENTIFIER: US 5440732 A
TITLE: Key-range locking with index trees

Brief Summary Text (24):
To compile this command, the query compiler 16 consults the database catalog, which contains the implementer's definitions of the data organization, such as that "INCOME" is a relation and INC and SSN are two of its attributes. It also determines the best manner in which to search for the indicated information, and it employs the index information in order to do this. If, for instance, the definer has required that an index ordered by social-security number be provided, then the query compiler produces a transaction routine that accesses the record by way of that index. In such a situation, the SSN attribute is the "key" by which the system finds the desired record (or records).

Current US Original Classification (1):
707/1

WEST

End of Result Set

Generate Collection Print

L22: Entry 5 of 5 File: USPT Jul 31, 1990

DOCUMENT-IDENTIFIER: US 4945475 A
TITLE: Hierarchical file system to provide cataloging and retrieval of data

Detailed Description Text (15):
If node 42 is an index node, then each record 60 and 61 is comprised of a key and pointer information. Further, NDFLINK 51 and NDBLINK 52 would contain adjacent index node linking pointers. If node 42 is a leaf node, then each record 60 and 61 is comprised of a key and data information. NDFLINK 51 and NDBLINK 52 would also contain leaf node linking pointers. It is also appreciated that although a particular format is illustrated for node 42, the format may be modified readily to include other types of information. Also, in the preferred embodiment data information in the leaf nodes of the HFS catalog B-Tree is used to address locations in memory where the actual data is stored.

Current US Original Classification (1):
707/1

WEST

Generate Collection | Print

L22: Entry 3 of 5 File: USPT Nov 9, 1993

DOCUMENT-IDENTIFIER: US 5261087 A
TITLE: Electronic information retrieval system with relative difference analyzer

Abstract Text (1):
A musical information retrieval system, for providing pieces of musical information to a user, is equipped with a relative difference analyzer for improvement of an average access time period even if the user is not familiar with an electronic processing system, and the relative difference analyzer comprises an input unit for providing a key word representative of a piece of identifying information indicative of an attribute of a musical composition, a data file including a plurality of books respectively having index records for storing respective pieces of musical attributive catalog information and detail data records for storing respective pieces of detailed musical data information associated with the pieces of detailed musical data information, respectively, an analyzer for searching the index records for candidates each partially identical with the piece of identifying information but partially different from the piece of identifying information, and a displaying unit for indicating the relative differences, if any.

Brief Summary Text (11):
In accordance with the present invention, there is provided an information retrieval system including a relative difference analyzer which comprises a) input means for providing a key word representative of a piece of identifying information, b) a data base including a data file having a plurality of books respectively having index records for storing respective pieces of attributive catalog information and detail data records for storing respective pieces of detailed data information, the pieces of attributive catalog information being associated with the pieces of detailed data information, respectively, c) analyzing means for searching the index records for relative differences of the piece of identifying information represented by the key word, each of the relative differences being partially identical with the piece of identifying information but partially different from the piece of identifying information, and d) indicating means for indicating the relative differences, if any.

Brief Summary Text (12):
In accordance with another aspect of the present invention, there is provided an information retrieval system for providing pieces of musical information including a relative difference analyzer, the relative difference analyzer comprising: a) input means for providing a key word representative of a piece of identifying information, the piece of identifying information being indicative of a musical attribute; b) a data base including a data file having a plurality of books respectively having index records for storing respective pieces of musical attributive catalog information and detail data records for storing respective pieces of detailed musical data information, the pieces of musical attributive catalog information being associated with the pieces of detailed musical data information, respectively, c) analyzing means for searching the index records for relative differences of the piece of identifying information represented by the key word, each of the relative differences being partially identical with the piece of identifying information but partially different from the piece of identifying information, and d) indicating means for indicating the relative differences, if any

Current US Original Classification (1):
707/3

CLAIMS:

1. A relative difference analyzer for an information retrieval system, said relative difference analyzer comprising:

a) input means for receiving a key word representative of a piece of identifying information, and for converting said key word into a key bit string having a predetermined fixed length;

b) storing means for storing a data base, said data base including a data file, said data file containing a plurality of books, each of said books including, respectively, index records for storing respective pieces of attributive catalog information and detail data records for storing respective pieces of detailed data information corresponding to said pieces of attributive catalog information, said pieces of attributive catalog information each being associated with a corresponding lock bit string, said lock bit strings being stored in said index records with said pieces of attributive catalog information, said lock bit strings each having a predetermined fixed length equal to the predetermined fixed length of said key bit string;

c) analyzing means for searching said index records for comparing said key bit string with said lock bit strings to determine a set of candidates from said pieces of attributive catalog information and for determining relative differences between said candidates and said piece of identifying information, each of said relative differences represented by a ratio between a first number and a second number, said first number representing a number of characters which are different between one of said candidates and said piece of identifying information, and said second number representing a total number of characters in said piece of identifying information;

d) indicating means for indicating said relative differences, if any; and

e) a bus system for electrically interconnecting said input means, said data base, said analyzing means and said indicating means.

2. A relative difference analyzer for a musical information system, said relative difference analyzer comprising:

a) input means for receiving a key word representative of a piece of identifying information, said piece of identifying information being indicative of a musical attribute, and for converting said key word into a key bit string having a predetermined fixed length;

b) storing means for storing a data base, said data base including a data film, said data file containing a plurality of books, each of said books including, respectively, index records for storing respective pieces of musical attributive catalog information and detail data records for storing respective pieces of detailed musical data information corresponding to said pieces of musical attributive catalog information, said pieces of musical attributive catalog information each being associated with a corresponding lock bit string, said lock bit strings being stored in said index records with said musical attributive catalog information, said lock bit strings having a predetermined fixed length equal to the predetermined fixed length of said key bit string;

c) analyzing means for searching said index records for comprising said key bit string with said lock bit strings to determine a set of candidates from said pieces of musical attributive catalog information and for determining relative differences between said candidates and said piece of identifying information each of said relative differences represented by a ratio between a first number and a second number, said first number representing a number of characters which are different between one of said candidates and said piece of identifying information, and said second number representing a total number of characters in said piece of identifying information;

d) indicating means for indicating said relative differences, if any; and

e) a bus system for electrically interconnecting said input means, said data base, said analyzing means and said indicating means,

L24: Entry 8 of 26 File: USPT Sep 21, 1999

DOCUMENT-IDENTIFIER: US 5956509 A
**** See image for Certificate of Correction ****
TITLE: System and method for performing remote requests with an on-line service network

Detailed Description Text (269):
The message layer 2000 of the MCP layer 210 uses the packet layer 2002 to provide a reliable byte stream between the client processors 102 and the Gateway 124. The packet layer 2002 handles error correction using CCITT CRC-32, and uses a sliding window protocol to handle flow control. To implement the sliding window protocol, each outbound packet contains an 8-bit packet sequence number. The sender increments the 8-bit packet sequence number with each successive packet transmission. Each packet also contains the 8-bit packet sequence number of the first non-received packet (with erroneous packets treated as non-received packets), so that packets sent in one direction serve as acknowledgements for packets sent in the opposite direction. Special acknowledgement packets (i.e., packets which do not contain client-server message data) are transmitted after a timeout period if the receiver has no data to send. Sliding window protocols are well known in the art, and are described in Tanenbaum, Computer Networks, 2nd Ed., Prentice-Hall, Inc., 1989, pp. 223-239.

Detailed Description Text (276):
Referring again to FIG. 21A, each packet 2100 further contains a four-byte cyclical redundancy check (CRC) code 2120 (in accordance with CRC-32) to permit detection and correction of errors. Each pet 2100 ends with a packet delimiter 2122, such as an ASCII carriage return code.